An improving dynamic programming algorithm to solve the shortest path problem with time windows

Nora Touati Moungla, Lucas Létocart, Anass Nagih

An improving dynamic programming algorithm to solve the shortest path problem with time windows

N. Touati Moungla[a], L. Létocart[b], A. Nagih[c]

[a] *LIX, École polytechnique, 91128 Palaiseau Cedex, France.*
[b] *LIPN, UMR 7030 CNRS, Institut Galilée - Université Paris 13, 99 avenue Jean-Baptiste Clément 93430 Villetaneuse, France.*
[c] *LITA, Université Paul Verlaine, Ile du Saulcy 57045 Metz Cedex 1, France.*

Abstract

An efficient use of dynamic programming requires a substantial reduction of the number of labels. We propose in this paper an efficient way of reducing the number of labels saved and dominance computing time. Our approach is validated by experiments on shortest path problem with time windows instances.

Keywords: Dynamic programming; Shortest path problem with time windows.

1. Introduction

Dynamic Programming (DP) (1) is a well-established method to solve shortest path problems. In the classical shortest path problem, the Ford-Bellman algorithm assigns a single label to each node representing the cost which is gradually improved. An extension of this approach was proposed in (2) in the shortest path problem with ressource constraints context. For the shortest path problem with time windows, the routes must be compared in terms of arrival times at the nodes as well as costs: at each node i a label (arrival time at node i, cost of the route) is defined. Several labels have to be stored at node i to calculate the labels of other nodes according to the following optimality principle: If X_{sj} is a minimum cost route from among all the routes from s to j arriving at j at time T_j or before, and if (i, j) is its terminal arc, then the sub-route X_{si} is a minimum cost route from among all routes from s to i arriving at i at time $T_j - t_{ij}$ or before, where t_{ij} is the travel time of arc (i, j).

This procedure can leads to the computation of a large number of labels at each node of the graph. Dominance rules are then used to compare subroutes arriving at a same location for discarding some of them. Dynamic programming algorithms (2; 3) rely on labels extension and a dominance procedure (4). In label setting algorithms, nodes are treated once and labels associated to a treated node are kept until the end of the resolution process. In label correcting approaches, nodes are repeatedly treated and their labels extended to all feasible directions.

The dominance procedure applied on each new label requires in the worst case, the exploration of all efficient labels of the current node. Since the number of labels increases exponentialy as a function of the problem's size, dominance computing time increases too. We propose in this work a new variant of the dynamic programming algorithm for accelerating the dominance procedure by reducing the number of labels saved and by identifying efficiently dominated labels.

We present in section 2 the Shortest Path Problem with Time Windows (SPPTW) and describe in section 3 its resolution by the dynamic programming algorithm. We present in section 4 a new variant of this method called dynamic programming with blocs. We validate in section 5 this new approach by experiments on shortest path problem with time windows instances. In section 6 we conclude.

2. Problem description

Consider a connected graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{A}$ is the set of arcs. With each arc $(i, j) \in \mathcal{A}$ is associated a real number cost c_{ij} and a non negative integer duration t_{ij}. Each node $i \in \mathcal{N}$ is characterized by a given time window $[a_i, b_i]$ within which the node may be visited. The SPPTW consists in finding the least cost route between two nodes $'s'$ and $'t'$ in the graph while visiting each node in the specified time window, it can be formulated as follows:

$$(SPPTW) \begin{cases} \min & \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} & \\ & \sum_{j\in\mathcal{V}} x_{sj} = \sum_{j\in\mathcal{V}} x_{jt} = 1 & \\ & \sum_{j\in\mathcal{V}} x_{ij} - \sum_{j\in\mathcal{V}} x_{ji} = 0, & \forall i \in \mathcal{N} \setminus \{s, t\} \\ & x_{ij} \in \{0, 1\}, & \forall (i, j) \in \mathcal{A} \\ & x_{ij}(T_i + t_{ij} - T_j) \leq 0, & \forall (i, j) \in \mathcal{A} \\ & T_j \in [a_j, b_j], & \forall j \in \mathcal{N} \\ & T_j \in \mathbb{N}, & \forall j \in \mathcal{N} \end{cases}$$

where $x_{ij}, (i,j) \in \mathcal{A}$ is the flow binary variable and $T_i, i \in \mathcal{N}$ is the visiting time of node i.

3. Dynamic programming algorithm

We associate to each l-th path X^l_{sj} from the source s to the node j, a (time, cost) label denoted by $E^l_j = (T^l_j, C^l_j)$, corresponding respectively to the duration and the cost of X^l_{sj}. This label is computed iteratively along the path $X^l_{sj} = (v_0, v_1, ..., v_q)$, where $v_0 = s$ and $v_q = j$ as follows:

$T_{v_0} = a_{v_0}$, $C_{v_0} = 0$
$T_{v_i} = \max\{a_{v_i}, T_{v_{i-1}+t_{v_{i-1}v_i}}\}, \ \ i = 1, ..., q$
$C_{v_i} = C_{v_{i-1}} + c_{v_{i-1}v_i}, \ \ i = 1, ..., q.$

A label E^l_j is feasible iff $T_{v_i} \leq b_{v_i}, \ \ \forall i = 1, ..., q$. Dominance rules are used to identify and discard useless labels.

Definition 1. *Let be X^1_{sj} and X^2_{sj} two different feasible paths from s to j and $E^1_j = (T^1_j, C^1_j)$, $E^2_j = (T^2_j, C^2_j)$ the associated labels respectively. Then, E^1_j dominates E^2_j iff $(T^2_j, C^2_j) \geq (T^1_j, C^1_j)$ and $(T^2_j, C^2_j) \neq (T^1_j, C^1_j)$. If E^1_j is not dominated by another label, it is called efficient.. The complete set of efficient labels characterize the Pareto optimal frontier (Figure 1).*

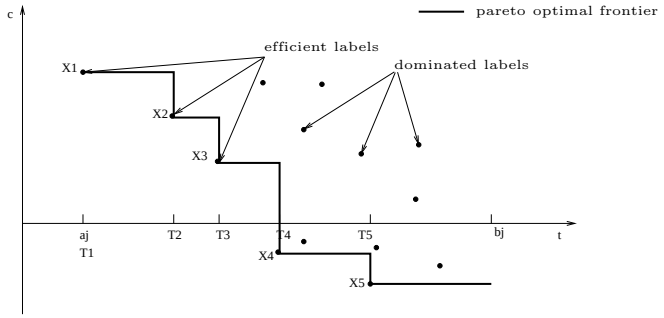

Figure 1: Pareto optimal frontier at node j

We denote by $\mathcal{E}_j = \{E^1_j, E^2_j, ..., E^p_j\}$ the set of efficient labels computed, and E^{p+1}_j a new label at node $j \in \mathcal{N}$. We present on Algorithm 1, the main steps of the dominance procedure applied on E^{p+1}_j.

Algorithm 1 Dominance operations applied on a new label in a dynamic programming procedure

next $\leftarrow$ true, $r \leftarrow 1$.
repeat
  if $E_j^{p+1} = E_j^r$ **then** $\mathcal{E}_j = \mathcal{E}_j \cup \{E_j^{p+1}\}$, next = false.
  if $E_j^{p+1} > E_j^r$ **then** next $\leftarrow$ false.
  if $E_j^{p+1} < E_j^r$ **then** $\mathcal{E}_j = \mathcal{E}_j \setminus \{E_j^r\}$, $r \leftarrow r + 1$.
  else $r \leftarrow r + 1$
until ((next = false) or $(r > p)$)
if next **then** $\mathcal{E}_j = \mathcal{E}_j \cup \{E_j^{p+1}\}$

The treatment of a great number of labels increases the computation time of the dominance procedure. We present in the next section an improvement method which the main goal is the reduction of the number of labels saved and the number of labels compared in the dominance procedure.

4. Dynamic programming with blocs

The principle of this method is the computation of spaces which contain only efficient labels. We associate to each node i a set of blocs, each bloc is defined by a triplet $B_i^k = (T_{i(low)}^k, T_{i(upp)}^k, C_{i(upp)}^k)$ where $T_{i(low)}^k$ corresponds to the lower bound on time, $T_{i(upp)}^k$ define the upper bound on time, and $C_{i(upp)}^k$ represents the upper bound on cost of the bloc indexed by k. We suppose that the blocs are not lower bounded by the cost.

4.1. Initialization

Let be i the treated node, the set of blocs is initialized by the bloc $B_i^0 = (a_i, b_i, +\infty)$ (figure 2). The bloc B_i^0 contains only feasible and efficient labels. Let be $E_i^1 = (C_i^1, T_i^1)$ a new label computed at node $i \in \mathcal{N}$ (Figure 3). Initially, none efficient label is saved at node i, E_i^1 is then efficient.

The new label E_i^1 allows to characterize the new bloc $B_i^1 = (T_i^1, b_i, C_i^1)$ (Figure 3), the bloc B_i^0 is then updated $B_i^0 = (a_i, T_i^1, +\infty)$.

4.2. Expansion and dominance

Proposition 1. *Let be $E_i^1, ..., E_i^l$ the set of efficient labels computed at node i and $\mathcal{B}_i = \{B_i^1, ..., B_i^l\}$ the associated blocs.. We consider $E_i = (C_i, T_i)$ a new*

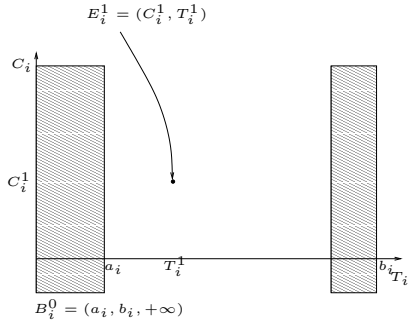

Figure 2: Initialization

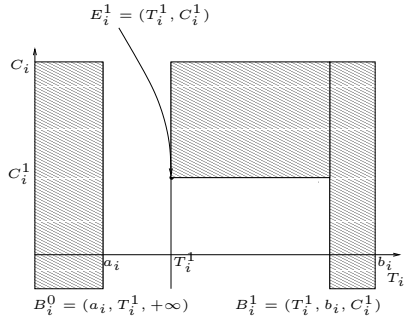

Figure 3: Label expansion

label computed at node $i \in \mathcal{N}$. If the label E_i belongs to a bloc $B_i^p, p = 0, ..., l$ (i.e.. $T^p_{i(low)} < T_i \leq T^p_{i(upp)}$ and $T_i < C^p_{i(upp)}$), then:

1. *E_i is a feasible label.*
2. *E_i is an efficient label.*
3. *The labels dominated by E_i are $E_i^{p+1}, ..., E_i^q$ which satisfy $C_i \leq C_i^{p+1}, ..., C_i \leq C_i^q$ respectively.*

proof 1.

1. *Each bloc in $\mathcal{B}_i \cup B_i^0$ belongs to the initialized bloc B_i^0 defined in section 4.1. All labels belonging to these blocs are then feasible.*
2. (a) *If E_i belongs to a bloc $B_i^p, p \in \{1, ..., l\}$. Suppose that E_i is dominated by the label $E_i^r, r \in \{1, ..., l\}$. Knowing that blocs (labels) are ordered by $T^k_{l(low)}$ increased, we deduce that the label E_i^r belongs to a bloc in the set $\{B_i^0, ..., B_i^{p-1}\}$.. On one hand we have $C_i^r \leq C_i$ (E_i^r dominates E_i) and $C_i \leq C_i^p$ (E_i belongs to the bloc B_i^p), so $C_i^r \leq C_i^p$, on another hand $T_i^r \leq T_i^p$, hence the contradiction (the labels E_i^r and E_i^p are efficient).*
   (b) *If E_i belongs to the bloc B_i^0. Knowing that it exists none label in the bloc B_i^0, all efficient labels of node i have durations greater than T_i, so the label E_i is efficient.*
3. *If $E_i \in B_i^p$, $p = 0, ..., l$, then $T_i \geq T_i^{p-1} > T_i^{p-2} ... > a_i$, we conclude that none label in the set $\{E_i^1, ..., E_i^{p-1}\}$ can be dominated by E_i. The labels which can be dominated by E_i belong to the set $\{E_i^{p+1}, ..., E_i^l\}$, more precisely, the labels dominated by E_i are $E_i^{p+1}, ..., E_i^q$ where $C_i \leq C_i^{p+1}, ..., C_i \leq C_i^q$ respectively.*

4.3. An illustrative example

Figure 4 presents an illustration of the dynamic programming with blocs on an example where $l = 6$. Let be $E_i^7 = (T_i^7, C_i^7)$ a new label computed at node i. Suppose that E_i^7 belongs to the bloc B_i^3, it is feasible and efficient. Labels dominated by E_i^7 belong to the set $\{E_i^4, E_i^5, E_i^6\}$ and satisfy $C_i^7 \leq C_i^q, q \in \{4, 5, 6\}$, i.e E_i^4. This label and the associated bloc B_i^4 are discarded, the bloc $B_i^7 = (T_i^7, T_i^5, C_i^7)$ is saved and the bloc B_i^3 is updated ($B_i^3 = (T_i^3, T_i^7, C_i^3)$).

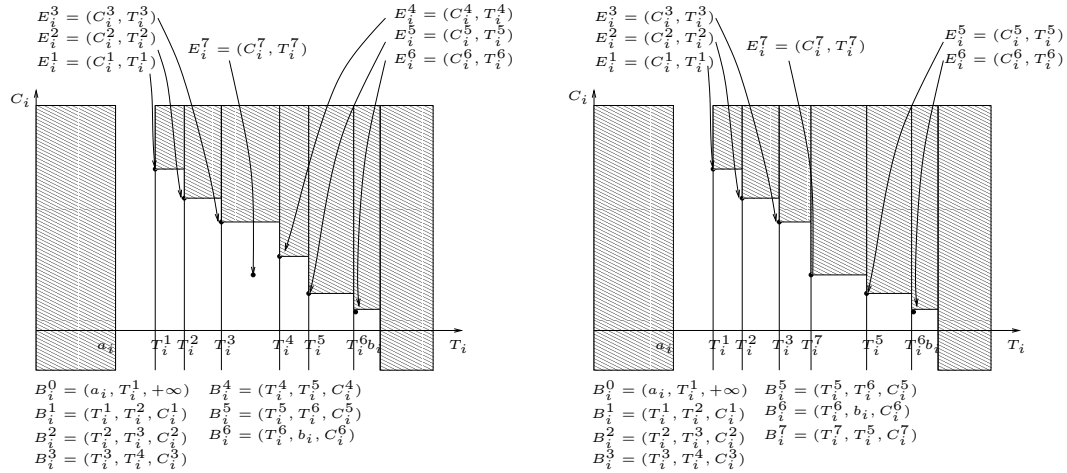

Figure 4: Illustrative example of expansion and dominance procedures

We denote by $\mathcal{B}_j = \{B_j^0, E_j^1, ..., E_j^l\}$ the set of blocs computed at node $j \in \mathcal{N}$, and E_j^{p+1} a new label at node j. We present on Algorithm 2, the main steps of the dominance procedure applied on E_j^{p+1} in our new procedure.

5. Experimentations

The results reported on Table 1 concern the resolution of the first pricing problem (SPPTW) in a column generation process for the resolution of the vehicle routing problem with time windows on acyclic graphs. The capacity constraints are relaxed. We consider 4 Solomon test instances ($C101\_25$ (25 costumers), $C101\_50$ (50 costumers), $C101$ (100 costumers) and $C1\_2\_1$ (200 costumers)) and 40 randomly generated instances. 10 instances are generated

Algorithm 2 Dominance operations applied on a new label in the dynamic programming with blocs procedure

if E_j^{p+1} belong to a bloc $B_j^s \in \mathcal{B}_j$ **then** $\mathcal{E}_j = \mathcal{E}_j \cup \{E_j^{p+1}\}$.
if $E_j^{p+1} \neq E_j^s$ **then** the bloc B_j^s is updated (section 4.3).
next $\leftarrow$ true.
$r \leftarrow s+1$.
repeat
  if $C_j^{p+1} <= E_j^r$ **then** $\mathcal{E}_j = \mathcal{E}_j \setminus \{E_j^r\}$, $\mathcal{B}_j = \mathcal{B}_j \setminus \{B_j^r\}$, $r \leftarrow r+1$.
  else next = false.
until ((next = false) or $(r > p)$)
$\mathcal{B}_j = \mathcal{B}_j \cup \{B_j^{p+1}\}$ (B_j^{p+1} is the bloc computed throw E_j^{p+1} (section 4.3)).

for each size: G_100 (100 costumers), G_120 (120 costumers), G_140 (140 costumers) and G_160 (160 costumers), all results reported for each randomly generated class size are average values over 10 test instances. All instances are solved by the label correcting algorithm 1.

Solomon	$C101\_25$			$C101\_50$		
instances	resolT	nbLab	nbLabDom	resolT	nbLab	nbLabDom
DP_LC	0,1”	343	680	3,3”	1 673	2 695
DP_Blocs	0.04”	79	4	0,2”	204	15
Solomon	$C101$			$C1\_2\_1$		
instances	resolT	nbLab	nbLabDom	resolT	nbLab	nbLabDom
DP_LC	72,0”	5 941	10 510	2 526,0”	39 922	1 946 876
DP_Blocs	3,2”	491	68	87,0”	1 036	11 425
Generated	G_100			G_120		
instances	resolT	nbLab	nbLabDom	resolT	nbLab	nbLabDom
DP_LC	186,0”	200 586	21 700 018	426,0”	335 421	116 011 504
DP_Blocs	18,9”	14 446	98 912	24,1”	92 325	123 413
Generated	G_140			G_160		
instances	resolT	nbLab	nbLabDom	resolT	nbLab	nbLabDom
DP_LC	462,0”	454 729	21 220 354	138,0”	105 478	11 836 750
DP_Blocs	27,0”	17 366	94 812	17,1”	57 758	115 377

resolT: resolution time.
nbLab: number of labels treated (labels on which the dominance procedure is applied).
nbLabDom: number of labels compared in the dominance procedure.

Table 1: Dynamic programming with blocs

Dynamic programming with blocs (DP_Blocs) decrease the number of labels treated by 79% compared to the label correcting algorithm (DP_LC), this is due to the reduction of the space search of new efficient labels using the blocs. DP_Blocs permits also to reducing efficiently the number of labels compared in the dominance procedure (more than 99%) compared to DP_LC. These results are predictable, as showed in proposition 2, dominance procedure is applied on a restricted number of labels. These results show the efficiency of the DP_Blocs which permits of decreasing the resolution time of the DP_LC method by 88%.

6. Conclusion

Traditionally used dominance rules in dynamic programming consists, after the expansion of a label, to verify if the new label is feasible and not dominated (in the opposite case, it is ignored) by comparing it with all efficient labels already computed. This procedure can be time consuming when the number of labels is large.

We propose in this paper an improving technique which objective is the reduction of the number of labels saved and the computation time of the dominance procedure. In the proposed approach feasibility and dominance tests are included in one, which consists to check if the treated label belongs to a bloc. In the case where the label belongs to a bloc, this method permits a direct access to dominated labels.

Experimentations on SPPTW instances reveal that this method reduces significantly the number of labels saved and the computation time of the dominance procedure.

[1] R. Bellman, Dynamic programming, Princeton university press, NJ, USA, 1957.

[2] M. Desrochers, An algorithm for the shortest path problem with resource constraints, Technical Report G-88-27,GERAD, 1988.

[3] S. Irnich and G. Desaulniers, Shortest path problems with resource constraints, In: Desaulniers et al. (eds), Column generation, Chap. 2. Springer, Berlin, 2005.

[4] M. Desrochers, F. Soumis, 1988. A generalized permanent labelling algorithm for the shortest path problem with time windows. INFOR 3, 191-212.